UNIVERSITETET
I OSLO

Relasjonsdatabasedesign

Funksjonelle avhengigheter

Oppdateringsanomalier

Dekomponering

Definisjon av nøkler

Gitt en relasjon R(A_1,A_2,…,A_n) **med tilhørende integritetsregler**.

La X være en delmengde av {A_1,A_2,…,A_n}.

Hvis t er et tuppel i en instans av R, betegner t[X] verdiene i t's X-attributter.

- **Supernøkkel** (superkey): En delmengde X av {A_1,A_2,…,A_n} som er slik at hvis t og u er to tupler hvor t≠u, så er t[X] ≠u[X].
  Dvs. t og u skal alltid ha forskjellig verdi i minst ett av attributtene i X
- **Kandidatnøkkel** (key): En minimal supernøkkel.
  Dvs: Fjerning av et hvilket som helst attributt fører til at de gjenværende attributtene ikke lenger utgjør en supernøkkel
- **Primærnøkkel** (primary key): En utvalgt blant kandidatnøklene.
  Alle relasjoner skal ha nøyaktig én primærnøkkel
- **Nøkkelattributt** (*prime attribute*): Attributt som er med i (minst) en kandidatnøkkel.

Kandidatnøkler (inklusive primærnøkler) benyttes til å uttrykke integritetsregler

Formell definisjon av funksjonell avhengighet

- Gitt en relasjon $R(A_1,A_2,\ldots,A_n)$ og la X, Y være delmengder av $\{A_1,A_2,\ldots,A_n\}$
- Y er **funksjonelt avhengig av** X hvis vi for enhver lovlig instans av R har at hvis instansen inneholder to tupler t_1 og t_2 hvor $t_1[X] = t_2[X]$, så må $t_1[Y] = t_2[Y]$
- I så fall skriver vi $X \rightarrow Y$

Funksjonell avhengighet (forts.)

- **FDer brukes til å uttrykke integritetsregler**
- Ofte snakker vi for korthets skyld om «FDen X → Y»
  (der **FD** står for Functional Dependency)
- Vi sier at «Y følger av X»,
  eller at «X bestemmer Y»

Funksjonell avhengighet og kandidatnøkler

- Merk at hvis X er en supernøkkel, så holder X → Y for enhver Y
- Omvendt: Hvis X → Y for enhver Y, så er X en supernøkkel
- Spesielt betyr dette at hvis vi angir at R (A_1,A_2,...,A_n) har en kandidatnøkkel X, betyr det at R har FDen X→ $A_1A_2...A_n$.
  - Hvis R($A_1A_2...A_n$) har en primærnøkkel X, betyr det at R har FDen X→ $A_1A_2...A_n$.

Ekvivalente mengder av FDer

La S, T være to mengder av FDer

- Definisjon:
  Vi sier at **S følger av T** hvis det er slik at enhver instans som oppfyller alle FDer i T, også oppfyller alle FDer i S
- Definisjon:
  Vi sier at **S og T er ekvivalente**
  hvis S følger av T og T følger av S

Slutningsregler for FDer

Hvert av regelsettene på de to neste lysarkene er

- Sunt: Vi kan ikke utlede noe som ikke gjelder
- Komplett: Alle FDer som kan vises å følge fra en mengde FDer ved bruk av definisjonen av FDer, kan også vises ved å bare bruke slutningsreglene

Hvert av regelsettene er minimale: Hvis vi fjerner en regel, er ikke det gjenværende regelsettet lenger komplett

Armstrongs slutningsregler

1. Refleksivitet:
   Hvis $Y \subseteq X$, så $X \rightarrow Y$

2. Utvidelse:
   Hvis $X \rightarrow Y$, så $XZ \rightarrow YZ$

3. Transitivitet:
   Hvis $X \rightarrow Y$ og $Y \rightarrow Z$, så $X \rightarrow Z$

Alternative regelsett

1. Refleksivitet:
   Hvis $Y \subseteq X$, så $X \rightarrow Y$

2. En av følgende:
   - Ekstensjon:
     Hvis $X \rightarrow Y$, så $X \rightarrow XY$
   - Union:
     Hvis $X \rightarrow Y$ og $X \rightarrow Z$, så $X \rightarrow YZ$

3. Transitivitet:
   Hvis $X \rightarrow Y$ og $Y \rightarrow Z$, så $X \rightarrow Z$

Trivielle FDer

- En FD som følger av refleksivitetsregelen

  «Hvis Y er en delmengde av X, så X → Y»

  kalles **triviell** fordi den er automatisk oppfylt

- En FD X → Y hvor Y – X ≠ Ø,

  kalles **ikketriviell**

Tillukningen av X mhp F

- Definisjon av **tillukningen av X mhp F**:
  La X være en mengde attributter og F en mengde FDer
  Da er tillukningen av X med hensyn på F lik mengden av attributter som er bestemt av X
- Notasjon: X^+ (mhp F)
- Egenskaper:
  - $X \rightarrow X^+$
  - Hvis A ikke er med i X^+, så har vi at A **ikke** er funksjonelt avhengig av X, dvs. $X \rightarrow A$ **holder ikke**
- Kjennetegnet på supernøkler (og derfor også kandidatnøkler) er at tillukningen av en supernøkkel inneholder **alle** attributtene i relasjonen

Tillukningsalgoritmen

Algoritme som beregner X^+ mhp. F:

1. T=X
2. Så lenge T forandres:
   A. Søk etter $Y \rightarrow Z$ i F hvor Y er en delmengde av T
   B. Hvis slik Y finnes: Legg Z inn i T (Dvs. Sett $T = T \cup Z$)
3. $X^+=T$

Tillukningsalgoritmen: Eksempel

- Gitt R(A,B,C,D,E,F) med FDene AB → C, BC → AD, D → E og CF → B
- Hva er $\{A,B\}^+$?

Test som avgjør om en FD $X \rightarrow Y$ følger fra en mengde FDer F

1. Beregn X^+ mhp. F

2. Hvis alle attributtene i Y er med i X^+, følger $X \rightarrow Y$ av F

   Hvis ikke, følger ikke $X \rightarrow Y$ av F

Hvordan finne alle kandidatnøkler I

Gitt en relasjon R(A_1,A_2, … ,A_n) med en mengde FDer F. Da er X en kandidatnøkkel i R hvis og bare hvis begge følgende punkter holder:

(a) X^+ er alle attributtene i R

(b) Uansett valg av en A_k i X er $(X- A_k)^+$ ikke alle attributtene i R.

Derfor bruker vi systematisk tillukning av attributtmengder (bottom-up) og sjekker hver mengde mot kriteriene (a) og (b) for å finne kandidatnøklene.

Hvordan finne alle kandidatnøkler II

For å begrense hvor mange tillukninger vi må beregne, bruker vi følgende observasjoner:

1. Hvis et attributt A ikke forekommer i noen høyreside i F, må A være med i alle kandidatnøklene.
2. Hvis et attributt A er med i minst én høyreside, men ingen venstresider, er A ikke med i noen av kandidatnøklene.

Hvordan finne alle kandidatnøkler III

Algoritme for å finne alle kandidatnøkler blir da:

1. Sett X lik mengden av attributter som ikke forekommer i noen høyreside i F.
2. Utvid systematisk X på alle mulige måter med attributter som forekommer i minst én venstreside i F. For hver slik X, beregn tillukningen X^+. Stans utvidelsen av en X hvis X^+ er samtlige attributter, dvs. X oppfyller kriterium (a). Hvis X også oppfyller kriterium (b), er X en kandidatnøkkel.

Eksempel

Gitt relasjonen R(A,B,C,D,E,F) med FDene AB → DE, C → A, BD → E, AE → BF.

Attributter som ikke forekommer i noen høyreside: C

Attributter som bare forekommer i høyresider: F

Begynn med C og utvid med A, B, D, E.

1. X := C. C^+ = CA. C er ikke en kandidatnøkkel.
2. Prøv å utvide X med B, D, E. (Siden A allerede er i C^+, er det ikke noe poeng å utvide C med A.)
   1. X:= BC. BC^+ = BCADEF. BC er en kandidatnøkkel.
   2. X:= CD. CD^+ = CDA. CD er ikke en kandidatnøkkel.
   3. X:= CE. CE^+ = BCADEF. CE er en kandidatnøkkel.

   Fortsett med X = CD. Prøv å utvide med B, E.
   4. X:= BCD. BCD^+ = BCADEF. Men BC er en kandidatnøkkel, så BCD oppfyller kriterium (a), men ikke (b).
   5. X:= CDE. CDE^+ = BCADEF. Men CE er en kandidatnøkkel, så CDE oppfyller kriterium (a), men ikke (b).

Hva kjennetegner god relasjonsdatabasedesign?

- Beslektet informasjon legges i samme relasjon
  - Ubeslektet informasjon legges *ikke* i samme relasjon
  - Brudd på dette vil ofte medføre dobbeltlagring av data og dermed forårsake ***oppdateringsanomalier***
- Så lite dobbeltlagring som mulig
  - Plassbehovet minimaliseres
  - Oppdatering forenkles
- Så få «glisne» relasjoner som mulig
  - Plassbehovet minimaliseres
  - Unngår problemer med håndtering av nil-verdier
- Korrekt totalinformasjon kan gjenskapes nøyaktig
  - Ingen falske data genereres

Eksempel: Varemagasindatabase

Bestilling(varenr, dato, varenavn, produsent, #bestilt)

Integritetsregler:

1) Et varenummer refererer til én bestemt vare
2) Utifra varenummeret er det gitt hvilken produsent det er snakk om
3) Det skal foretas maksimalt én bestilling av en gitt vare på en gitt dato

Varemagasindatabasen versjon 1

Bestilling(varenr, dato, varenavn, produsent, #bestilt)

Integritetsregler:

1) varenr → varenavn
2) varenr → produsent

} varenr → varenavn, produsent

3) varenr, dato → #bestilt

Eksempelinstans Bestilling

Bestilling				
varenr	**dato**	**varenavn**	**produsent**	**#bestilt**
1	d1	A	a	3
2	d1	B	b	8
1	d2	A	a	2

Sekundær informasjon

«varenavn» og «produsent» er eksempler på ***sekundær informasjon***. Dette er viktig informasjon, men den hører ikke hjemme i en tabell over bestillingene; det er mer naturlig å ha en egen tabell som har oversikt over varer og produsenter.

Oppdateringsanomalier

- **Innsettingsanomalier**
  - Opprettholde konsistente verdier
  - Håndtere sekundær informasjon
  - Håndtere nil i kandidat- og fremmednøkler
- **Slettingsanomalier**
  - Unngå tap av sekundær informasjon
- **Modifiseringsanomalier**
  - Opprettholde konsistente verdier
  - Oppdatere sekundær informasjon

Hvordan unngå oppdateringsanomalier

- Unngå/fjern oppdateringsanomalier og dobbeltlagring ved å splitte (dekomponere) relasjonene slik at dobbeltlagring blir borte!

Dekomposisjon av en relasjon

- Gitt en relasjon $R(A_1,A_2,\ldots,A_n)$

  En ***dekomposisjon*** $D = \{R_1,R_2,\ldots,R_m\}$ av R er en samling med relasjoner $R_1,R_2,\ldots,R_m$ som er slik at følgende to krav holder:

  – alle attributtene i hver R_k er også attributt i R

  – samtlige attributter $A_1,A_2,\ldots,A_n$ kan gjenfinnes i minst en av relasjonene $R_1,R_2,\ldots,R_m$

Dekomposisjon: Ønskede egenskaper

- Eliminasjon av oppdateringsanomalier
- Rekonstruksjon av den opprinnelige instansen er mulig
  - Dekomposisjonen er **tapsfri**, dvs. at naturlig join aldri vil gi falske tupler
- Bevaring av funksjonelle avhengigheter

Eksempel: Varemagasindatabasen versjon 2

Vare(varenr, varenavn, produsent)

Ordre(varenr, dato, #bestilt)

Integritetsregler:

1) varenr → varenavn, produsent
2) varenr, dato → #bestilt
3) varenr i Ordre er fremmednøkkel til Vare

Eksempelinstans
Vare, Ordre

Gammel modell

Den nye modellens instanser fås ved å projisere den gamle modellens instans på attributtene i hver av de nye relasjonene

Ny modell

Bestilling				
varenr	**dato**	**varenavn**	**produsent**	**#bestilt**
1	d1	A	a	3
2	d1	B	b	8
1	d2	A	a	2

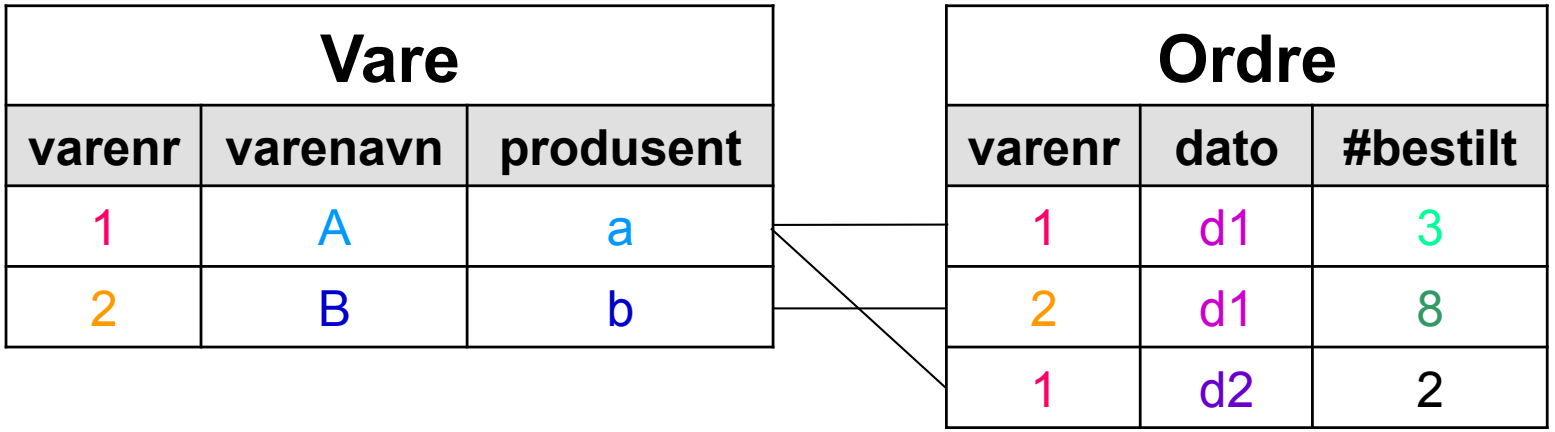
Vare		
varenr	**varenavn**	**produsent**
1	A	a
2	B	b

Ordre		
varenr	**dato**	**#bestilt**
1	d1	3
2	d1	8
1	d2	2

Naturlig join

- Vi ønsker å kunne rekonstruere den opprinnelige instansen
- ***Naturlig join*** er en operasjon der to relasjoner slås sammen til en ved at to og to tupler pares hvis og bare hvis de har like verdier i attributter med likt navn

Vare ⋈ Ordre



Vare		
varenr	**varenavn**	**produsent**
1	A	a
2	B	b

Ordre		
varenr	**dato**	**#bestilt**
1	d1	3
2	d1	8
1	d2	2

Vare ⋈ Ordre				
varenr	**varenavn**	**produsent**	**dato**	**#bestilt**
1	A	a	d1	3
2	B	b	d1	8
1	A	a	d2	2

Eksempel: Varemagasindatabasen versjon 3

Vare(varenr, varenavn, produsent)

Datobestilt(varenr, dato)

Antallbestilt(varenr, #bestilt)

Integritetsregler:

1) varenr → varenavn, produsent
2) varenr, dato → #bestilt
3) varenr i Datobestilt er fremmednøkkel til Vare
4) varenr i Antallbestilt er fremmednøkkel til Vare

Eksempelinstanser Datobestilt, Antallbestilt

Ordre		
varenr	**dato**	**#bestilt**
1	d1	3
2	d1	8
1	d2	2

Datobestilt	
varenr	**dato**
1	d1
2	d1
1	d2

Antallbestilt	
varenr	**#bestilt**
1	3
2	8
1	2

⋈

Datobestilt ⋈ Antallbestilt		
varenr	**dato**	**#bestilt**
1	d1	3
~~1~~	~~d1~~	~~2~~
2	d1	8
~~1~~	~~d2~~	~~3~~
1	d2	2

Naturlig join på de to tabellene gir flere tupler enn i den opprinnelige tabellen! = **Falske tupler**

Retningslinjer for dekomposisjon av relasjoner

- **Motivasjon**: Fjerne oppdateringsanomalier
- **Teknikk**: Dekomponere problemrelasjoner
- **Betingelse**: Opprinnelig instans skal alltid kunne rekonstrueres nøyaktig ved naturlig join – dekomposisjonen er **tapsfri**
- **Problemer**:
  - Hvordan vurdere objektivt om en samling relasjoner er god/dårlig?
  - Hvordan sikre at en dekomposisjon aldri gir falske tupler (er tapsfri)?

Chasealgoritmen

Gitt en dekomposisjon av R(A,B,...) til relasjonene S_1, ... , S_k og et sett F med FDer for R. Er dekomposisjonen tapsfri?

1. Lag en tabell med en kolonne for hvert attributt i R og en rad for hver S_i
2. I kolonnen for attributt A, for hver rad i:
   - Skriv a hvis A er et attributt i S_i
   - Skriv a_i hvis A ikke er et attributt i S_i
3. Så lenge det skjer forandringer i tabellen og det ikke finnes en rad uten subskriptverdier:
   - For hver FD $X \rightarrow Y$, for alle rader i tabellen med lik X-verdi, gjør Y-verdiene like. Hvis en av Y-ene er en verdi uten subskript, skal denne velges

- Hvis en rad er uten subskriptverdier, er dekomposisjonen tapsfri, ellers ikke.

Eksempel

- R(A,B,C,D,E)
- FDer: A → C, B → C, C → D, DE → C, CE → A
- Dekomposisjon: AD, AB, BE, CDE, AE

Pkt 3 i chase-algoritmen skal anvendes på følgende tabell:

	A	**B**	**C**	**D**	**E**
AD	a	b_1	c_1	d	e_1
AB	a	b	c_2	d_2	e_2
BE	a_3	b	c_3	d_3	e
CDE	a_4	b_4	c	d	e
AE	a	b_5	c_5	d_5	e

Hvordan dekomponere tapsfritt

Fagins teorem

Gitt en relasjon R(XYZ) med FDer F.
En dekomposisjon D={XY, XZ} er tapsfri mhp. F hvis og bare hvis minst en av følgende holder:

1) $X \rightarrow Y \in F^+$
2) $X \rightarrow Z \in F^+$

Her er F^+ mengden av alle FDer som kan avledes av F

Eksempel: R(A,B,C,D), F={C→AD}
Dekomposisjon: R_1(A,C,D), R_2(B,C)

Normalformer

- Normalformer er et uttrykk for hvor godt vi har lykkes i en dekomposisjon
- Jo høyere normalform, jo færre oppdateringsanomalier
- Det finnes algoritmer for å omforme fra lavere til høyere normalformer

Ekstramateriale

Slutningsregler
(fra Armstrongs slutningsregler og de alternative regelsettene)

1. Refleksivitet (begge regelsettene):
   Hvis $Y \subseteq X$, så $X \rightarrow Y$
2. Utvidelse (bare Armstrong):
   Hvis $X \rightarrow Y$, så $XZ \rightarrow YZ$
3. Ekstensjon (bare alternative regler; da trengs ikke union):
   Hvis $X \rightarrow Y$, så $X \rightarrow XY$
4. Union (bare alternative regler; da trengs ikke ekstensjon):
   Hvis $X \rightarrow Y$ og $X \rightarrow Z$, så $X \rightarrow YZ$
5. Transitivitet (begge regelsettene):
   Hvis $X \rightarrow Y$ og $Y \rightarrow Z$, så $X \rightarrow Z$

Bevis for utvidelsesregelen:
Hvis X → Y, så XZ → YZ

- Anta at X → Y, og at det finnes to tupler t og u slik at t[XZ] = u[XZ]
- Da har vi spesielt at t[X] = u[X] og at t[Z] = u[Z]
- Siden X → Y, og t[X] = u[X], er t[Y] = u[Y]
- Da har vi både at t[Y] = u[Y], og at t[Z] = u[Z]
- Sammen gir det at t[YZ] = u[YZ]
- Dermed har vi bevist at XZ → YZ holder

Bevis for den transitive regelen:
Hvis X → Y og Y → Z, så X → Z

- Anta at X → Y, at Y → Z, og at det finnes to tupler t og u slik at t[X] = u[X]
- Siden X → Y, og t[X] = u[X], er t[Y] = u[Y]
- Siden Y → Z, og t[Y] = u[Y], er t[Z] = u[Z]
- Dermed har vi bevist at X → Z holder